Travaux pratiques Services Web SOAP
Environnement de travail

Les services seront déployés sous Glassfish, sous la forme d'une archive war (WEB-INF mais sans fichier web.xml, et répertoire classes qui contient les .class des services déployés).

Vous placerez le(s) classes représentant les services dans un package.

L'utilitaire wsgen (à utiliser après avoir compilé le service) permet de générer les fichiers qui permettront la communication avec le service Web déployé

```
wsgen -cp WEB-INF/classes:$CLASSPATH -d WEB-INF/classes -wsdl ServiceSOAP
```

où *ServiceSOAP* est le nom de la classe implémentant le service.

L'archive web (war) que vous déployez contient les fichiers précédemment générés

```
jar cvf WSSOAP.war WEB-INF/* (avec éventuellement index.html)
```

Pour consulter le WSDL après déploiement

```
localhost:8080/WSSOAP/ServiceSOAPService?WSDL
```

Notez l'ajout de **Service** après le nom de votre classe !

Pour tester le service

```
localhost:8080/WSSOAP/ServiceSOAPService?Tester
```

Rappel pour déployer en ligne de commandes :

```
asadmin deploy --force NomArchive.war
```

Pour communiquer avec le service via un client C#, du côté du client on génère les classes avec l'outil wsdl du framework .NET:

```
wsdl http://NomMachineServeur:NumPortHTTP/WSSOAP/ServiceSOAPService?wsdl
```

Pour compiler le client

```
mcs ClientWS.cs ServiceSOAPService.cs -r:System.Web.Services
```

Pour communiquer avec le service via un client Java, du côté du client on génère les classes avec l'outil `wsimport` :

```
wsimport http://localhost:8080/WSSOAP/ServiceSOAPService?WSDL
```

Vous pouvez utiliser l'option –keep si vous souhaitez conserver les .java générés. Sinon ils sont automatiquement supprimés pour nettoyer le répertoire.

TP1 – Production et consommation de services SOAP

Question 1 – Premier service SOAP : le traditionnel "Salut" !
Créez (avec un éditeur de texte basique) un service auquel on transmet une chaîne de caractères et qui retourne un message de salutation. Consultez le fichier WSDL associé au service via le navigateur, utilisez le service dans le navigateur.

Question 2 – Consommation du service en Java – Première version
Ecrivez une première version du client en utilisant la création d'un représentant du service en transmettant son URL et un qualified name (pour faire la correspondance entre l'URL et le service). Sur ce service, obtenez le port pour utiliser la méthode proposée. Vous aurez copié chez le client le fichier compilé qui correspond à l'interface uniquement.

Question 3 – Consommation du service en Java – Seconde version
Utilisez maintenant l'outil `wsimport` pour générer les classes chez le client, et reprenez le client pour y déclarer un objet IBonjour. Obtenez le port pour utiliser la méthode proposée.

Question 4 – Consommation du service en C#
Ecrivez un client C# qui déclare un objet BonjourService et qui appelle la méthode proposée. Vous aurez à générer les fichiers nécessaires au client avec wsdl.

Question 5 – Consommation de service sur le net
Ecrivez une page jsp (à inclure dans le war) qui propose un formulaire de saisie du nom et qui appelle ensuite le service (implémentation) pour afficher le message de salutation.

Question 6 – Consommation de service sur le net
Retrouvez les services de TextCasing proposés sur le net, testez-en un en ligne et ensuite à travers un client C# puis un client Java avec l'annotation @WebServiceRef. L'exécution du client Java se fait en utilisant `appclient` *ClientTextCase.*

Question 7 – Consommation de service sur le net
A partir du site www.webservicex.net, consultez les définitions wsdl du service qui renvoie la ville à partir d'un code postal (par exemple 02106) des USA (USA Zip code Information), ou de celui qui permet d'obtenir le pays à partir d'une adresse IP (GeoIPService).
Testez dans un navigateur, puis utilisez dans un client C#. Utilisez la classes `System.Xml.XmlNode` et les propriétés `FirstChild` et `InnerText` pour accéder aux données.
Pour essayer en Java : `http://stackoverflow.com/questions/9080259`

TP2 – Authentification

Question 1

On va modifier le service Bonjour pour qu'il n'accepte que des personnes préalablement identifiées. Les informations d'authentification seront transmises dans les headers HTTP.

Dans le service, on utilise une ressource qui représente le contexte sur laquelle on peut récupérer les HTTP_REQUEST_HEADERS. On obtient une Map qui associe une chaîne de caractères à la collection de valeurs (String), qu'il faut parcourir pour trouver les informations d'authentification.

Constatez que le service ne fonctionne plus dans un navigateur.

Question 2

Ecrire une application Java cliente qui construit les headers et les affecte au contexte puis appelle le service.

Question 3

Proposez une page JSP qui permet l'authentification et qui enchaîne sur la salutation.

TP 3 – Gestion des exceptions

Question 1
Modifier le service Bonjour pour qu'il lève une exception si la chaîne transmise est le mot "erreur". Testez dans le navigateur.

Question 2
Ecrire une application cliente qui capte l'erreur et affiche le message correspondant.

Vous utiliserez la commande

```
java -Dcom.sun.xml.ws.transport.http.client.HttpTransportPipe.dump=true Client
```

pour voir l'échange des messages SOAP.

Question 3
Modifiez le service précédent pour qu'il lève une Fault SOAP en utilisant l'annotation @WebFault. Constatez dans le navigateur.

Plus de détails sur ces 2 liens :

http://java.globinch.com/enterprise-java/web-services/jax-ws/jax-ws-exceptions-faults-annotation-exception-and-fault-handling-examples/

http://io.typepad.com/eben_hewitt_on_java/2009/07/using-soap-faults-and-exceptions-in-java-jaxws-web-services.html

Question 4
Modifiez l'application cliente pour qu'elle affiche les détails de l'exception lancée, et visualisez le message SOAP reçu.

Question 5 – Travail subsidiaire (si vous avez du temps)
Vous pouvez écrire un programme qui génère une requête SOAP et qui affiche une réponse SOAP en vous inspirant du code proposé à cette adresse :

http://stackoverflow.com/questions/19291283/soap-request-to-webservice-with-java

TP 4 – Service Stateful et gestion des sessions

Pour illustrer l'utilisation de session, vous travaillerez sur un service très simple qui incrémente un compteur à chaque fois que le service est invoqué. Ce compteur sera conservé comme un attribut de la session.

Question 1
Dans un premier temps, écrivez un service Stateful qui gère le compteur et qui propose une unique méthode getCompteur. Ajoutez un second service qui sera le gestionnaire du premier service. Vous pouvez vous inspirer des informations sur la page :
https://blogs.oracle.com/sujit/entry/ws_addressing_and_stateful_webservice

Question 2
Ecrivez une application cliente qui invoque plusieurs fois le service et constatez la valeur du compteur. Vous constatez que plusieurs clients (lourd ou navigateur) en parallèle ne partagent pas le compteur (en temporisant dans l'application par exemple en attente d'une entrée de l'utilisateur).

Question 3
On va maintenant gérer le même compteur à travers un session, qui se contente d'incrémenter un compteur, pas d'ouverture de session ni de création de l'objet compteur. Ce service reste dans la portée de la session grâce à l'annotation @HttpSessionScope.

Question 4
Ecrivez une application cliente qui démarre la session et invoque plusieurs fois le service et constatez la valeur du compteur. Vous constatez que plusieurs clients (lourd ou navigateur) en parallèle travaillent dans leur propre session (en temporisant dans l'application par exemple en attente d'une entrée de l'utilisateur).

Question 5
On va maintenant gérer le même compteur à travers un session récupérée à travers le contexte. Ecrivez un service qui retrouve une session qui serait démarrée par un client et qui lève une erreur si il n'en existe pas. Le compteur est géré comme attribut de session, qu'il faut

éventuellement créer si il n'existe pas, ou mettre à jour en l'incrémentant.

http://stackoverflow.com/questions/8036827/how-can-i-manage-users-sessions-when-i-use-web-services

Ajoutez un second service qui retrouve une session qui serait démarrée et qui accède à l'attribut de session. On suppose que ce second service sera toujours appelé après le premier, inutile de gérer les erreurs. Faites attention à ce que les 2 méthodes web ne portent pas le même nom, lors de la génération, les fichiers correspondant à leurs classes se remplaceraient mutuellement.

Question 6

Complétez l'application cliente de façon à ce qu'elle utilise les deux services. Constatez que les 2 services ne sont toujours pas dans la même session.

Question 7

Ajoutez dans chacun des services une seconde méthode qui retourne la liste des cookies de chaque service.

Question 8

Constatez dans l'application cliente que les numéros de session sont différents. Constatez aussi avec 2 instances du même service.

Pour la lecture des cookies :

http://tugdualgrall.blogspot.fr/2007/02/oracle-web-services-sharing-session.html

Question 8

En reprenant des idées à partir de cette page

https://developer.jboss.org/thread/155259?tstart=0&_sscc=t

proposez une solution pour une gestion de session entre plusieurs services.